Intro to Git: Understanding Git commands

Git is similar to a cloud service, but one that saves every version of your work. It helps you keep track of changes, back up your work, and collaborate with others.

For example, imagine two students are working on different parts of a data cleaning project: one is handling data for Virginia (VA), and the other for North Carolina (NC). In Git, they can each work on their own "branch". Later, they can merge their work into the main project, where their advisor can review all changes in one place.

Another example: if a student makes a change to their code but later realizes it caused an error, Git allows them to revert to an earlier version easily. There's no need to manually undo their changes or re-upload files—Git keeps a record of every saved version for quick recovery.

🌱 Overview: Git's Workflow Structure

Thinking of Git as having three key areas:

- **Working Directory**: Where you make changes (edit/add/delete files).
- **Staging Area**: Where you prepare files for uploading.
- **Local Repository**: Your project's file on your computer..
    - **To create a new repository**: git init [name-of-repo] — This sets up a new environment for Git to track the content in the folder.
    - **To check if Git is running correctly**: git status — This will show if everything is running properly and the current status for reference.
- **Remote Repository**: A copy of your project stored on a cloud (GitHub**)**, similar to Google Drive.

---

git add

- **Function**: Selects the modified files that you want to upload to the main branch.
- **Usage**:
    - git add <filename> to add the specific files.
    - git add . to add all changes
- **Effect**: Moves modified files from the **working directory** to the **staging area, or in other words, "preparing stage."**

- **Analogy**: Like putting papers into a folder before submitting them. You're saying, "These are ready."

---

git commit

- **Function**: Records the changes you've made into Git's version history.
- **Usage**:
    - `git commit -m "Your message here"`
        - For example, git commit -m "Adding a Readme."
- **Analogy**: Once your documents are organized, you submit the folder to the office. In return, you receive a stamped receipt acknowledging the submission.

---

git log

- **Function**: Displays a list of all your past commits.
- **Usage**:
    - `git log`
- **Effect**: Shows commit messages, timestamps, and IDs.

---

git push

- **Function**: Sends your local commits to the remote repository (e.g., GitHub).
- **Usage**:
    - git push
- **Effect**: Copies changes from the **local repository** to the **remote repository**.
- **Analogy**: Like uploading your journal to the cloud — it's now backed up and shareable.

Additional Git Commands for Exploring and Reverting Changes

git log Variations

- git log — View the full history of commits with details.
    - git log -3 — Show only the three most recent commits.
    - git log [filename] —Show the commit history of a specific file.
    - git log --since='Apr 2 2024' — Show commits made since a specific date.
    - git log --since='Apr 2 2024' --until='Apr 10 2024' — Show commits between two dates.

git show

- git show [commit-hash] — Display the changes, author, and message for a specific commit.

git diff

- git diff — Show the difference between the working directory and the staging area.

    - git diff --staged [filename] — Compare the staged version of a file to the last committed version.
    - git diff [hash1] [hash2] — Compare two specific commits.
    - git diff HEAD~1 HEAD — Compare the last commit with the current (latest) commit.

Reverting and Undoing Changes

- git revert — Create a new commit that undoes the changes from a previous commit.
    - git revert --no-edit HEAD — Revert the most recent commit without opening an editor.
    - git revert -n HEAD — Revert the last commit without committing immediately.
    - git checkout -- [filename] — Discard changes in the working directory and restore the last committed version of a specific file.
    - git restore --staged [filename] — Unstage a file (move it from the staging area back to the working directory).